Importing Dependencies:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

Data collection

In [2]:
```
wine_data = pd.read_csv('winequality-red.csv')
```

In [3]:
```
wine_data
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

In [4]:
```
wine_data.isna().sum()
```

Out[4]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

Data analysis and Visualization

In [5]:
```
wine_data.columns
```

Out[5]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [6]:
```
wine_data.groupby('quality').mean()
```

Out[6]:

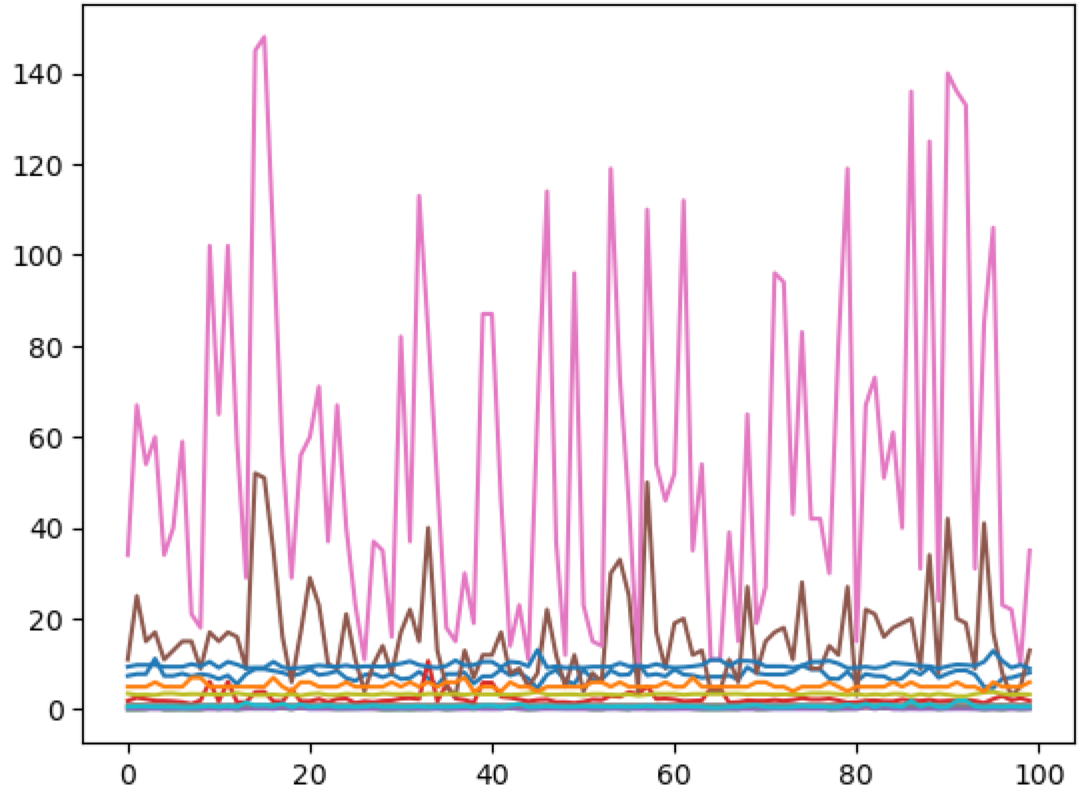
quality	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
3	8.360000	0.884500	0.171000	2.635000	0.122500	11.000000	24.900000	0.997464	3.398000	0.570000	9.955000
4	7.779245	0.693962	0.174151	2.694340	0.090679	12.264151	36.245283	0.996542	3.381509	0.596415	10.265094
5	8.167254	0.577041	0.243686	2.528855	0.092736	16.983847	56.513950	0.997104	3.304949	0.620969	9.899706
6	8.347179	0.497484	0.273824	2.477194	0.084956	15.711599	40.869906	0.996615	3.318072	0.675329	10.629519
7	8.872362	0.403920	0.375176	2.720603	0.076588	14.045226	35.020101	0.996104	3.290754	0.741256	11.465913
8	8.566667	0.423333	0.391111	2.577778	0.068444	13.277778	33.444444	0.995212	3.267222	0.767778	12.094444

In [7]:
```
wine_data.shape
```

Out[7]:
```
(1599, 12)
```

In [8]:

```
plt.plot(wine_data[: 100]);
```



In [9]:
```
wine_data.describe()
```

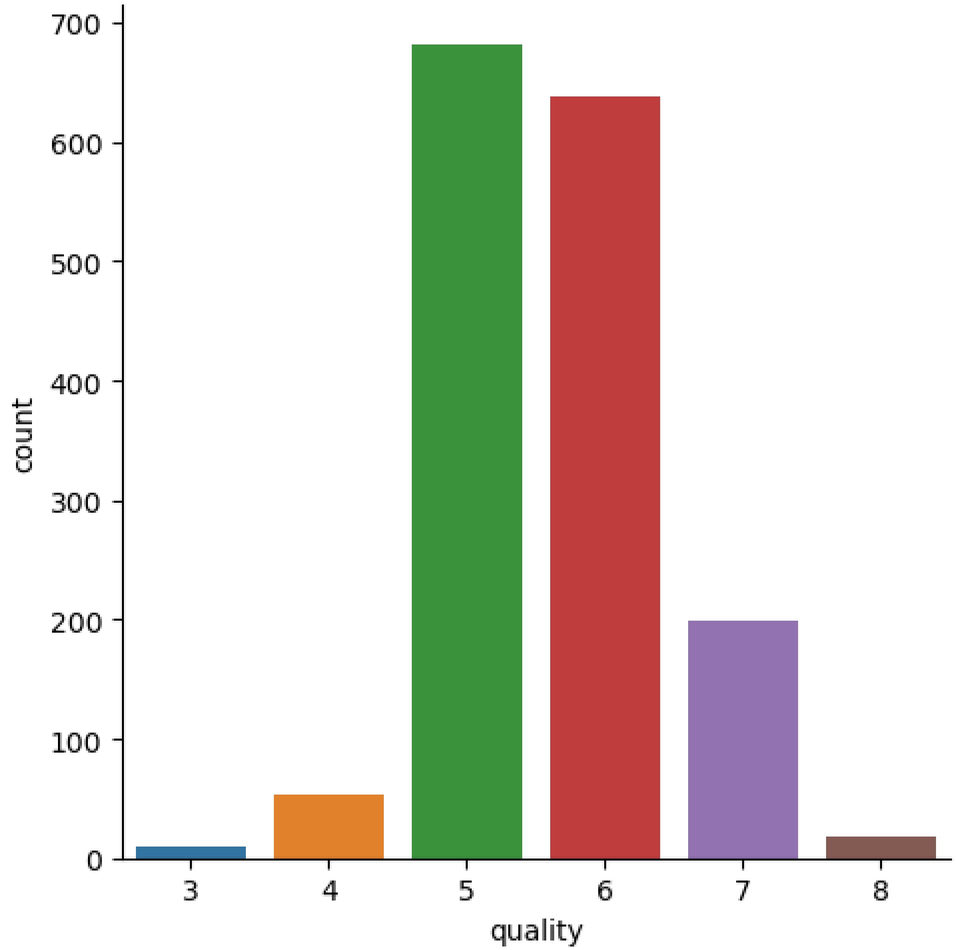
Out[9]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	s
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2

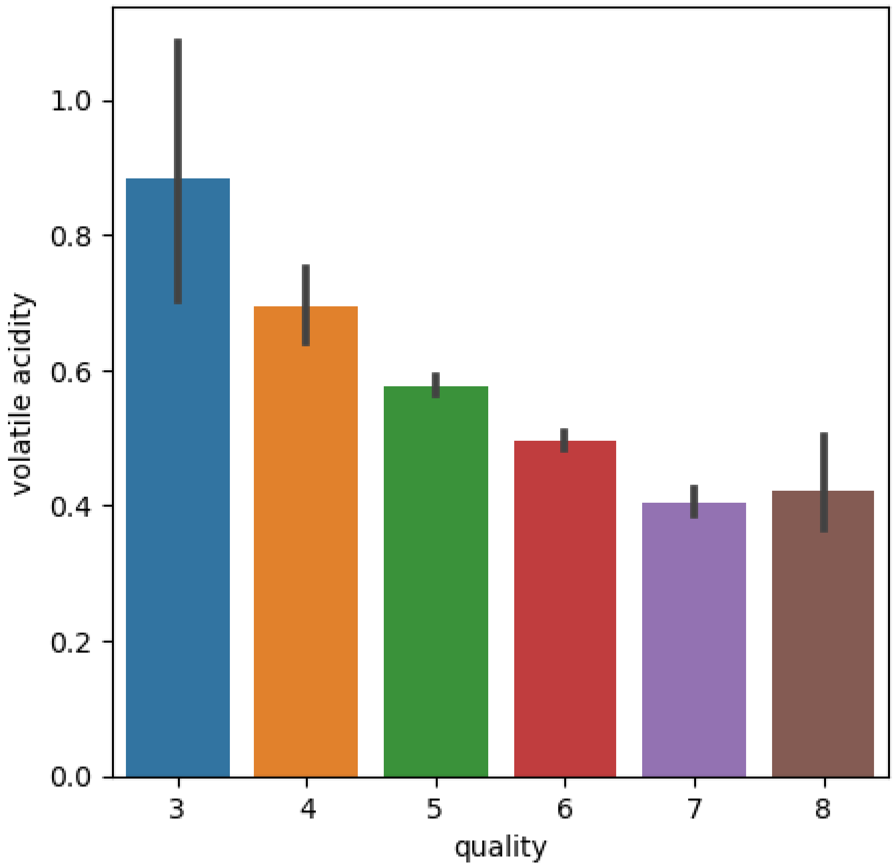
In [10]:
```
wine_data['quality'].value_counts()
```

Out[10]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

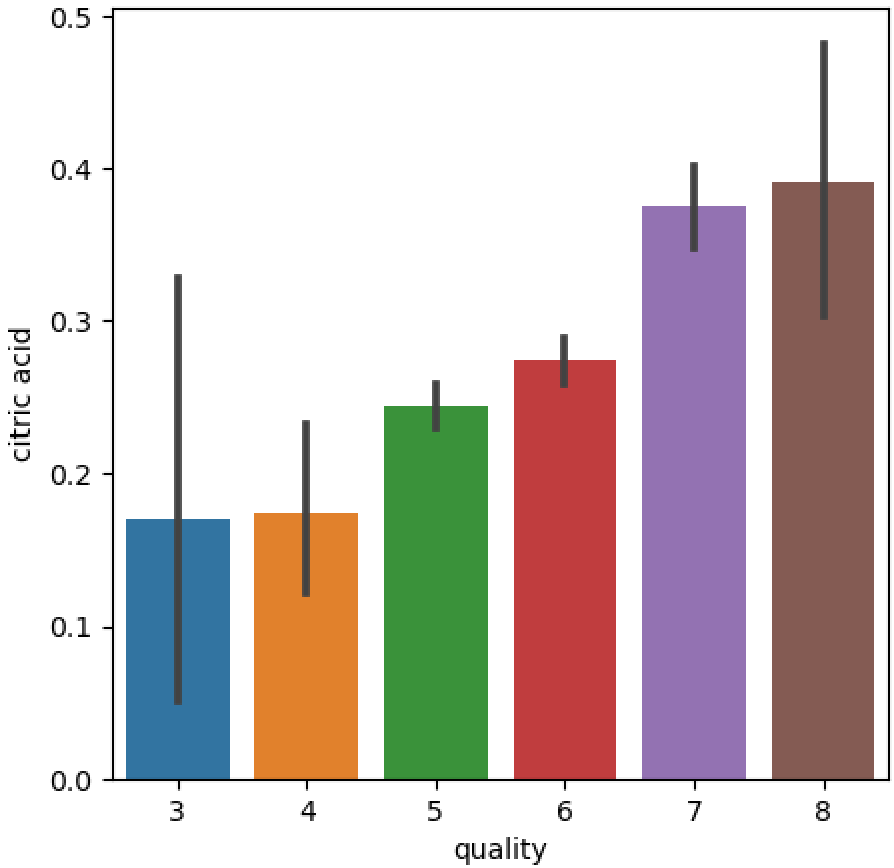
In [11]:
```
# number of values for each quality

sns.catplot(x = 'quality', data = wine_data, kind='count');
```



In [12]:
```
# volatile acidity vs quality
plot = plt.figure(figsize= (5, 5))
sns.barplot(x = 'quality', y = 'volatile acidity', data = wine_data);
```



In [13]:
```
# citric acid vs quality
plot = plt.figure(figsize= (5, 5))
sns.barplot(x = 'quality', y = 'citric acid', data = wine_data);
```



Correlation of all the columns.

Types of correlation

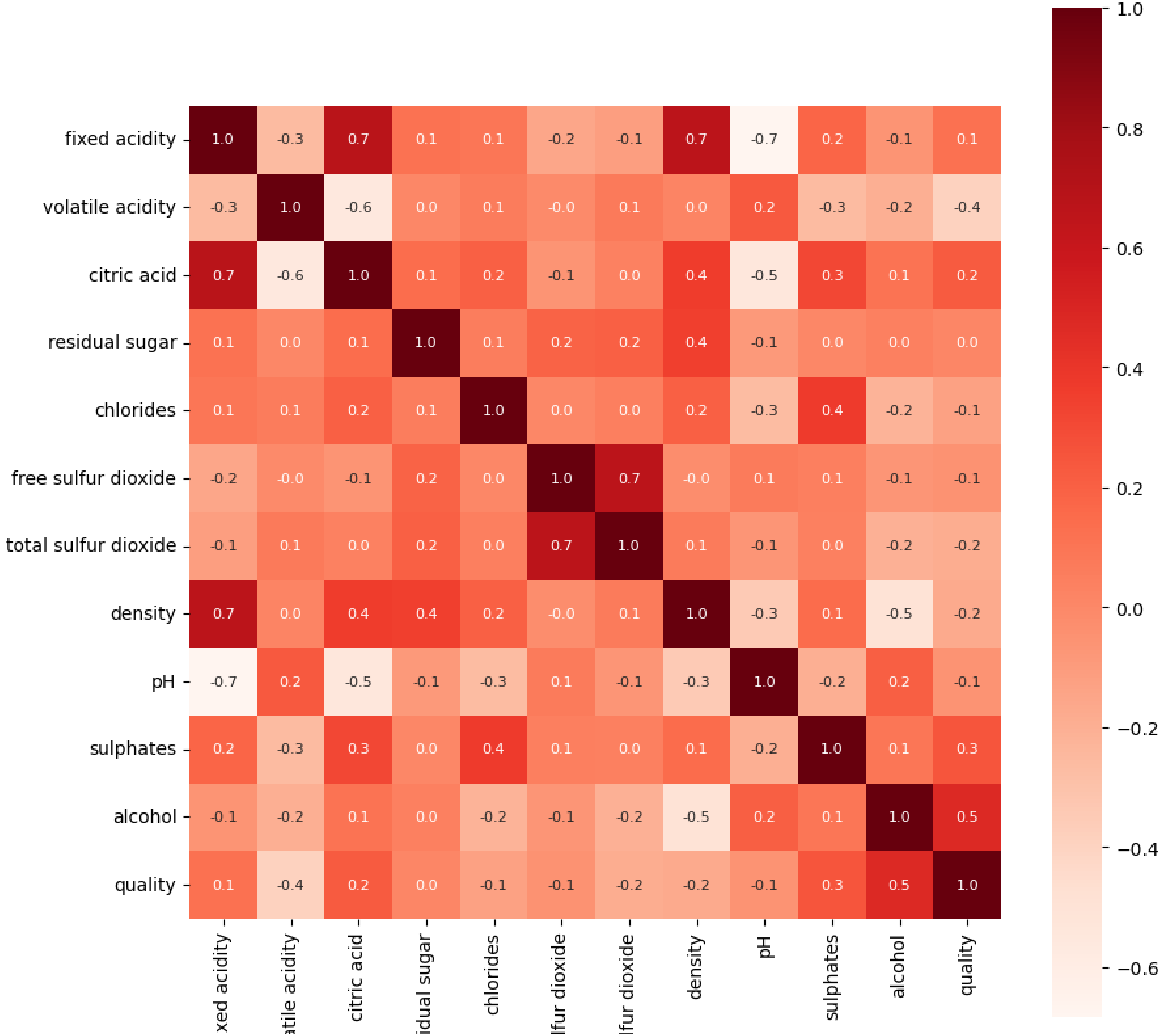
1. Positive correlation
2. Negative correlation
3. zero correlation

In [14]:
```
correlation = wine_data.corr()
correlation
```

Out[14]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.668047	-0.682978	0.183006	-0.061668	0.1
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.022026	0.234937	-0.260987	-0.202288	-0.3
citric acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	0.364947	-0.541904	0.312770	0.109903	0.2
residual sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	0.355283	-0.085652	0.005527	0.042075	0.0
chlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	0.200632	-0.265026	0.371260	-0.221141	-0.1
free sulfur dioxide	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000	0.667666	-0.021946	0.070377	0.051658	-0.069408	-0.0
total sulfur dioxide	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666	1.000000	0.071269	-0.066495	0.042947	-0.205654	-0.1
density	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946	0.071269	1.000000	-0.341699	0.148506	-0.496180	-0.1
pH	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377	-0.066495	-0.341699	1.000000	-0.196648	0.205633	-0.0
sulphates	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658	0.042947	0.148506	-0.196648	1.000000	0.093595	0.2
alcohol	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408	-0.205654	-0.496180	0.205633	0.093595	1.000000	0.4
quality	0.124052	-0.390558	0.226373	0.013732	-0.128907	-0.050656	-0.185100	-0.174919	-0.057731	0.251397	0.476166	1.0

In [15]:
```
# heatmap to understand the correlation between the columns

plt.figure(figsize=(10, 10))
sns.heatmap(correlation, cbar =True, square = True, fmt='.1f', annot=True, annot_kws={'size':8}, cmap
```


fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
quality
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

fi vola resi free sul total sul

separating the data and label

In [54]:
```
x = wine_data.drop('quality', axis = 1)
y = wine_data['quality']
```

In [55]:
```
x.shape, y.shape
```

Out[55]: ((1599, 11), (1599,))

In [56]:
```
x
```

Out[56]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0

1599 rows × 11 columns

In [57]:
```
y
```

Out[57]:
```
0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1599, dtype: int64
```

Label Binarization

In [58]:
```
y = wine_data['quality'].apply(lambda y_value: 1 if y_value >= 7 else 0)
```

In [59]:
```
y
```

Out[59]:
```
0       0
1       0
2       0
3       0
4       0
       ..
1594    0
1595    0
1596    0
1597    0
1598    0
Name: quality, Length: 1599, dtype: int64
```

Train and Test split

In [60]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, stratify=y, random_state=42)
```

In [61]:
```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[61]:
```
((1279, 11), (320, 11), (1279,), (320,))
```

Model Training

In [62]:
```
model = RandomForestClassifier()
```

In [63]:
```
model
```

Out[63]:
```
RandomForestClassifier()
```

In [64]:
```
model.fit(x_train, y_train)
```

Out[64]:
```
RandomForestClassifier()
```

Model Evaluation

In [65]:
```
model.score(x_train, y_train)
```

Out[65]:
```
1.0
```

In [66]:

```
y_preds = model.predict(x_test)
y_preds
```

Out[66]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], dtype=int64)
```

In [67]:

```
from sklearn.metrics import confusion_matrix

# Assuming you have the true labels for the test dataset in y_true
confusion = confusion_matrix(y_test, y_preds)
print(confusion)

```

```
[[275   2]
 [ 17  26]]
```

In [68]:

```
accuracy_score(y_test, y_preds)
```

Out[68]:

```
0.940625
```

In [90]:
```
from sklearn.metrics import classification_report

class_report = classification_report(y_test, y_preds)
print("Classification Report:")
print(class_report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       277
           1       0.93      0.63      0.75        43

    accuracy                           0.94       320
   macro avg       0.94      0.81      0.86       320
weighted avg       0.94      0.94      0.94       320
```

Hyperparameter Tunning

In [81]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],    # Number of trees in the forest
    'max_depth': [None, 5, 10],          # Maximum depth of each tree
    'min_samples_split': [2, 5, 10],     # Minimum number of samples required to split a node
    'min_samples_leaf': [1, 2, 4]        # Minimum number of samples required at each leaf node
}
```

In [82]:
```
# Create the RandomForestClassifier model
model = RandomForestClassifier()

# Perform grid search
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(x_train, y_train)
```

Out[82]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [None, 5, 10],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [50, 100, 200]})
```

In [83]:
```
# Get the best hyperparameters and score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Hyperparameters:", best_params)
print("Best Score:", best_score)
```

```
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best Score: 0.9022702205882354
```

There is a decrease in model score

In [84]:
```
from sklearn.ensemble import RandomForestClassifier

# Define the best parameters
best_params = {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}

# Create a new instance of the RandomForestClassifier model with the best parameters
model = RandomForestClassifier(**best_params)

# Fit the model to the training data
model.fit(x_train, y_train)

```

Out[84]:
```
RandomForestClassifier(n_estimators=300)
```

In [85]:
```
y_preds = model.predict(x_test)
y_preds
```

Out[85]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], dtype=int64)
```

In [86]:
```
from sklearn.metrics import confusion_matrix

# Assuming you have the true labels for the test dataset in y_true
confusion = confusion_matrix(y_test, y_preds)
print(confusion)

```

```
[[275   2]
 [ 16  27]]
```

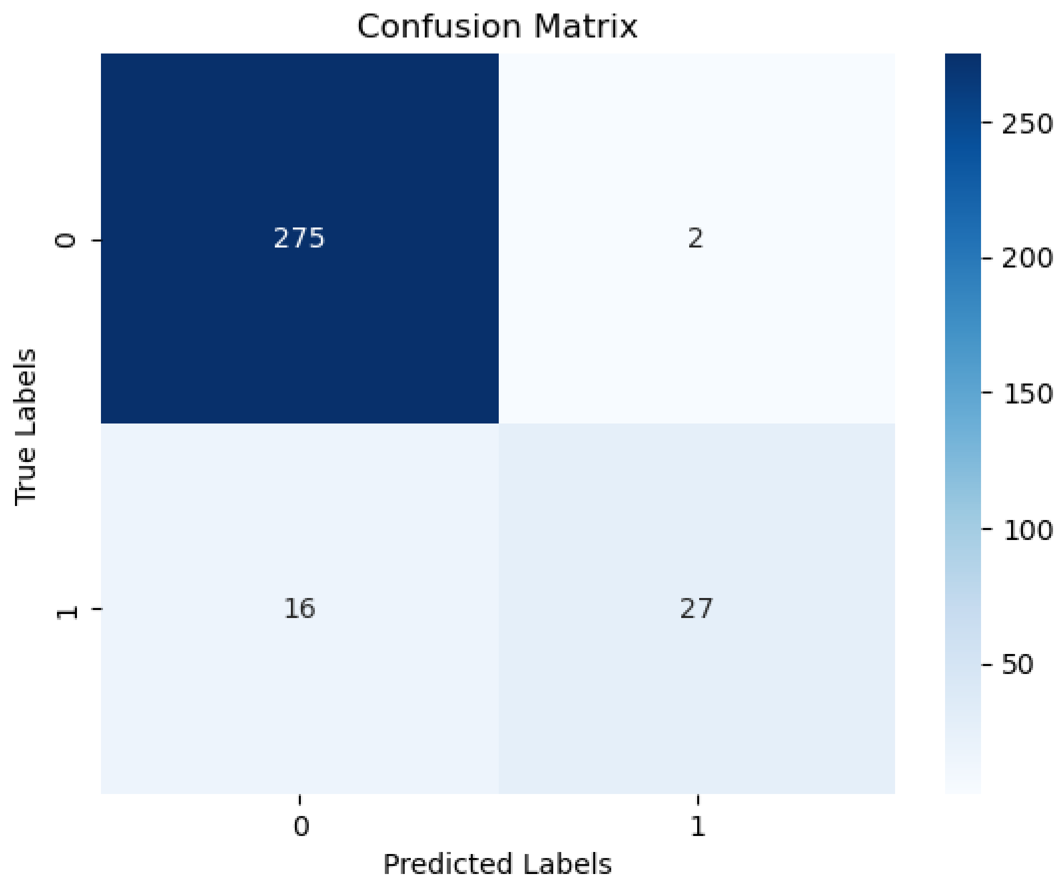
In [89]:

```
# Create a DataFrame with the actual and predicted values
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_preds})

# Print the DataFrame
print(df)
```

```
      Actual  Predicted
434        0          0
213        0          0
1447       0          0
231        0          0
763        0          0
...      ...        ...
306        0          0
839        0          0
166        0          0
1505       0          0
580        0          0

[320 rows x 2 columns]
```

In [88]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have the confusion matrix stored in the 'confusion' variable

# Create a heatmap using seaborn and format annotations as integers
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues')

# Add labels, title, and axis ticks
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')

# Show the plot
plt.show()

```



Building a predictive system

Accuracy on training data

In [107]:

```
x_training_prediction = model.predict(x_train)
training_data_accuracy = accuracy_score(x_training_prediction, y_train)
print('Accuracy on training data : ', training_data_accuracy)
```

```
Accuracy on training data :  1.0
```

Accuracy on test data

In [108]:
```
x_test_prediction = model.predict(x_test)
test_data_accuracy = accuracy_score(x_test_prediction, y_test)
print('Accuracy on training data : ', test_data_accuracy)
```

```
Accuracy on training data :  0.94375
```

In [109]:
```
x
```

Out[109]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0

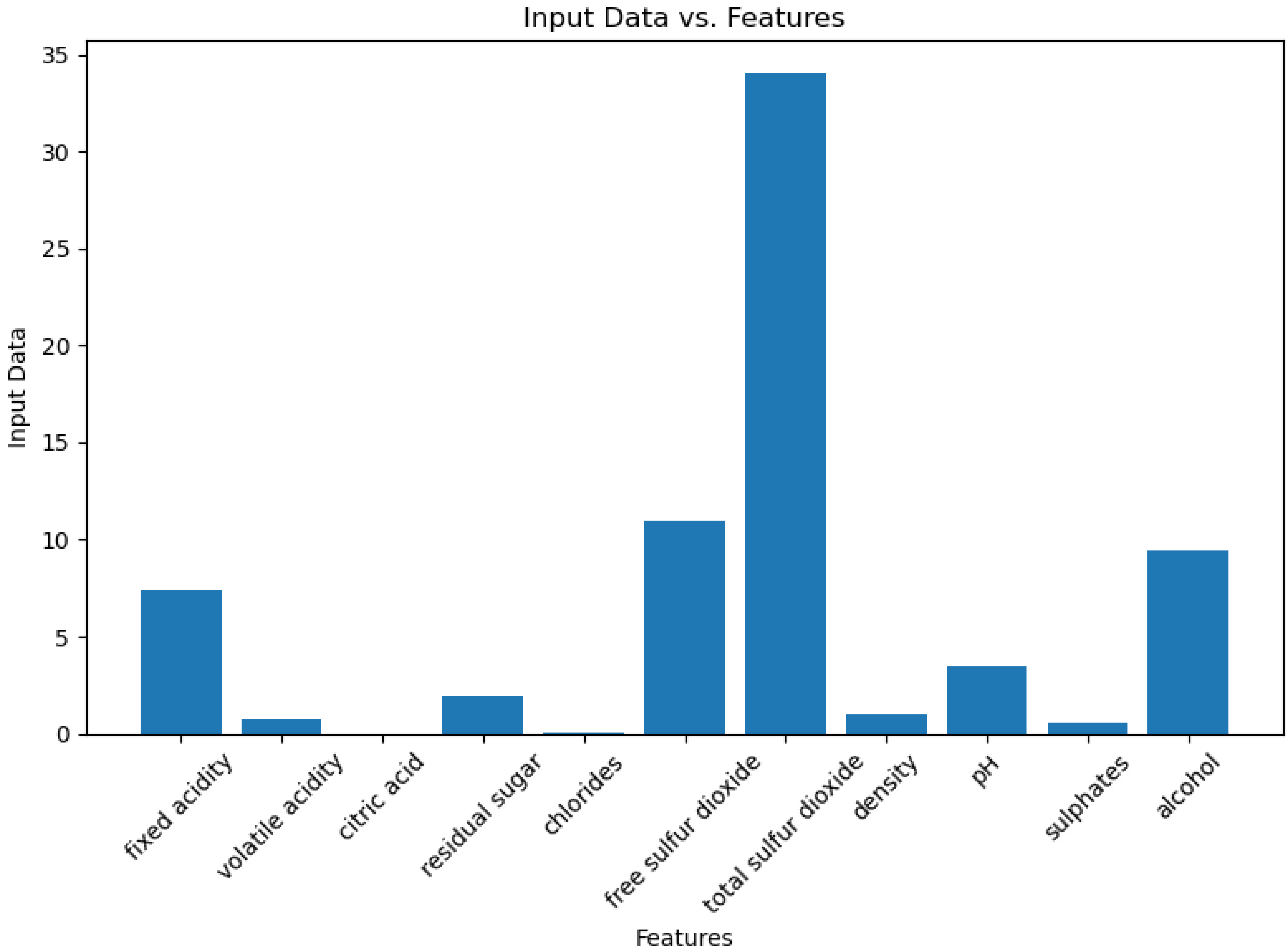
1599 rows × 11 columns

In [112]:

```
import numpy as np

input_data = (7.4, 0.7, 0.0, 1.9, 0.076, 11.0, 34.0, 0.9978, 3.51, 0.56, 9.4)
feature_names = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', '

# Change the input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the np array as we are predicting for one instance
input_data_reshape = input_data_as_numpy_array.reshape(1, -1)

# Assign feature names to the reshaped array
input_data_with_features = pd.DataFrame(input_data_reshape, columns=feature_names)

prediction = model.predict(input_data_with_features)
if prediction[0] == 1:
    print('Good Quality Wine')
else:
    print('Bad Quality Wine')

```

```
Bad Quality Wine
```

In [113]:
```
# Plot the input data against the features
plt.figure(figsize=(8, 6))
plt.bar(feature_names, input_data_as_numpy_array)
plt.xlabel('Features')
plt.ylabel('Input Data')
plt.title('Input Data vs. Features')
plt.xticks(rotation=45)  # Rotate the x-axis labels by 45 degrees
plt.tight_layout()  # Adjust the layout to prevent label overlapping
plt.show()
```

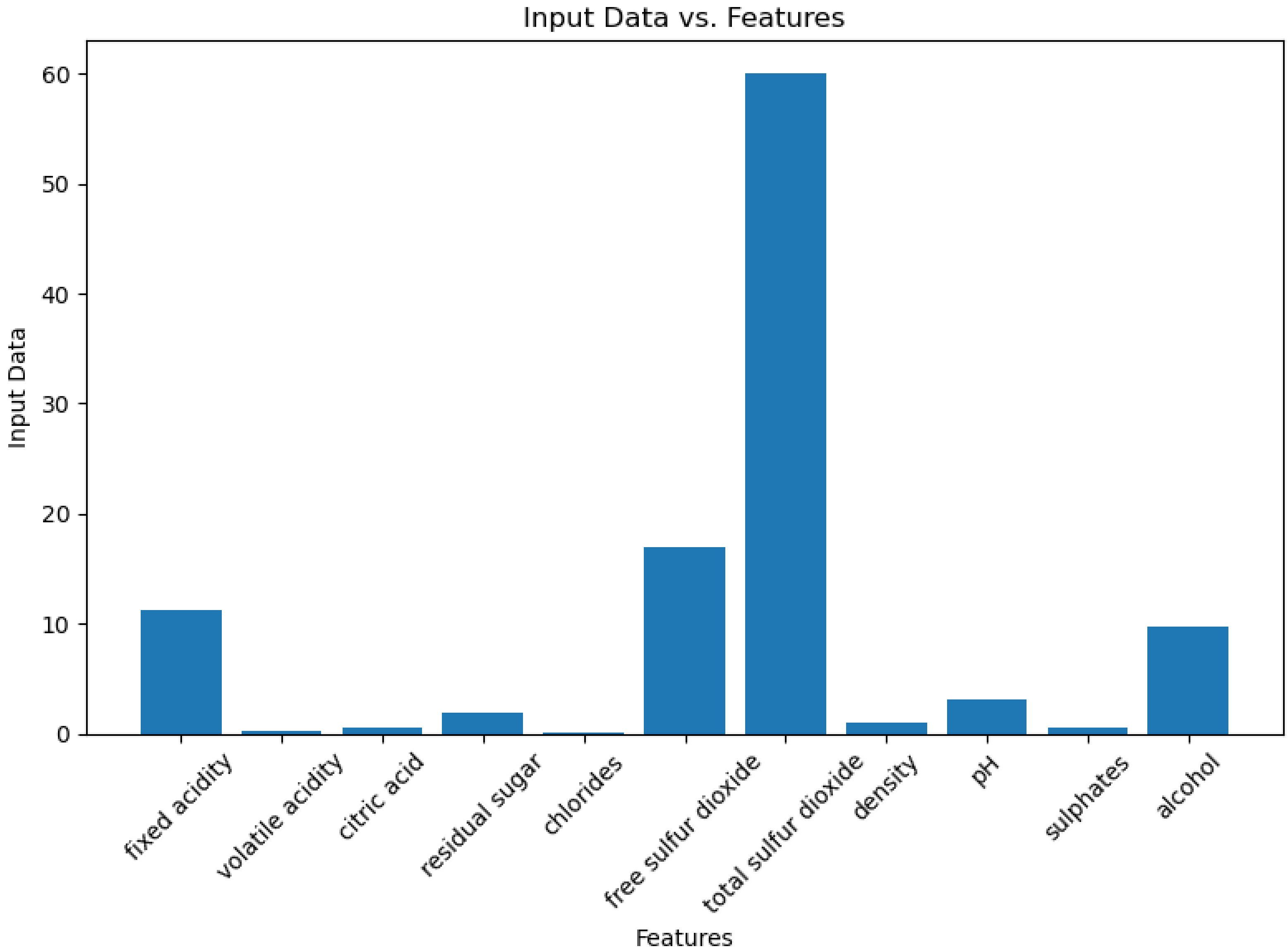

Input Data vs. Features
Input Data
35
30
25
20
15
10
5
0
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
Features

In [114]:

```
import numpy as np

input_data = (11.2,0.28,0.56,1.9,0.075,17.0,60.0,0.998,3.16,0.58,9.8)
feature_names = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', '

# Change the input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the np array as we are predicting for one instance
input_data_reshape = input_data_as_numpy_array.reshape(1, -1)

# Assign feature names to the reshaped array
input_data_with_features = pd.DataFrame(input_data_reshape, columns=feature_names)

prediction = model.predict(input_data_with_features)
if prediction[0] == 1:
    print('Good Quality Wine')
else:
    print('Bad Quality Wine')

```

```
Bad Quality Wine
```

In [115]:
```
# Plot the input data against the features
plt.figure(figsize=(8, 6))
plt.bar(feature_names, input_data_as_numpy_array)
plt.xlabel('Features')
plt.ylabel('Input Data')
plt.title('Input Data vs. Features')
plt.xticks(rotation=45)  # Rotate the x-axis labels by 45 degrees
plt.tight_layout()  # Adjust the layout to prevent label overlapping
plt.show()
```


Input Data vs. Features
Input Data
0
10
20
30
40
50
60
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
Features

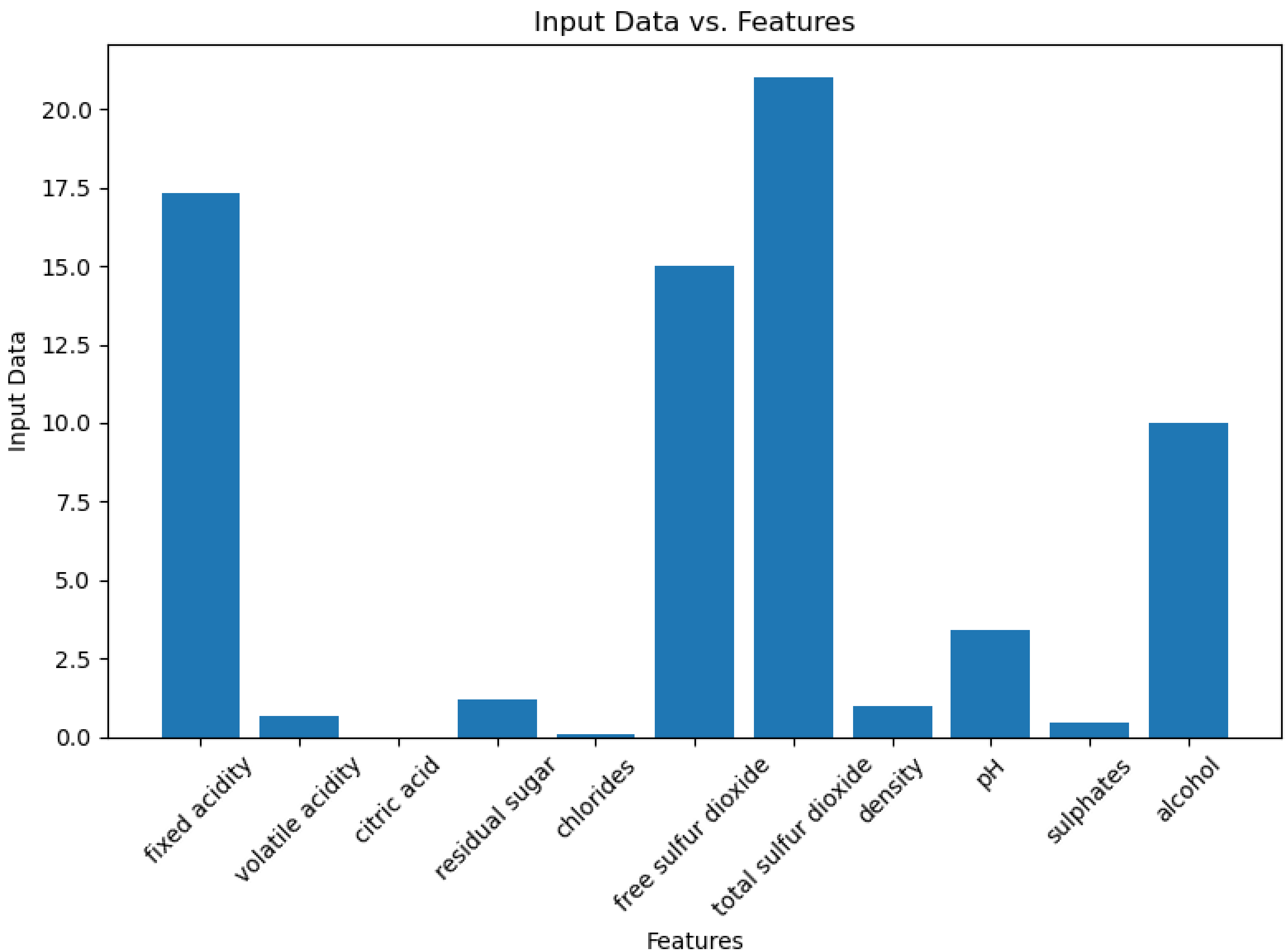
In [116]:

```
import numpy as np

input_data = (17.3,0.65,0.0,1.2,0.065,15.0,21.0,0.9946,3.39,0.47,10.0)
feature_names = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', '

# Change the input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the np array as we are predicting for one instance
input_data_reshape = input_data_as_numpy_array.reshape(1, -1)

# Assign feature names to the reshaped array
input_data_with_features = pd.DataFrame(input_data_reshape, columns=feature_names)

prediction = model.predict(input_data_with_features)
if prediction[0] == 1:
    print('Good Quality Wine')
else:
    print('Bad Quality Wine')

```

```
Good Quality Wine
```

In [117]:

```
# Plot the input data against the features
plt.figure(figsize=(8, 6))
plt.bar(feature_names, input_data_as_numpy_array)
plt.xlabel('Features')
plt.ylabel('Input Data')
plt.title('Input Data vs. Features')
plt.xticks(rotation=45)  # Rotate the x-axis labels by 45 degrees
plt.tight_layout()  # Adjust the layout to prevent label overlapping
plt.show()
```


Input Data vs. Features
Input Data
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
Features

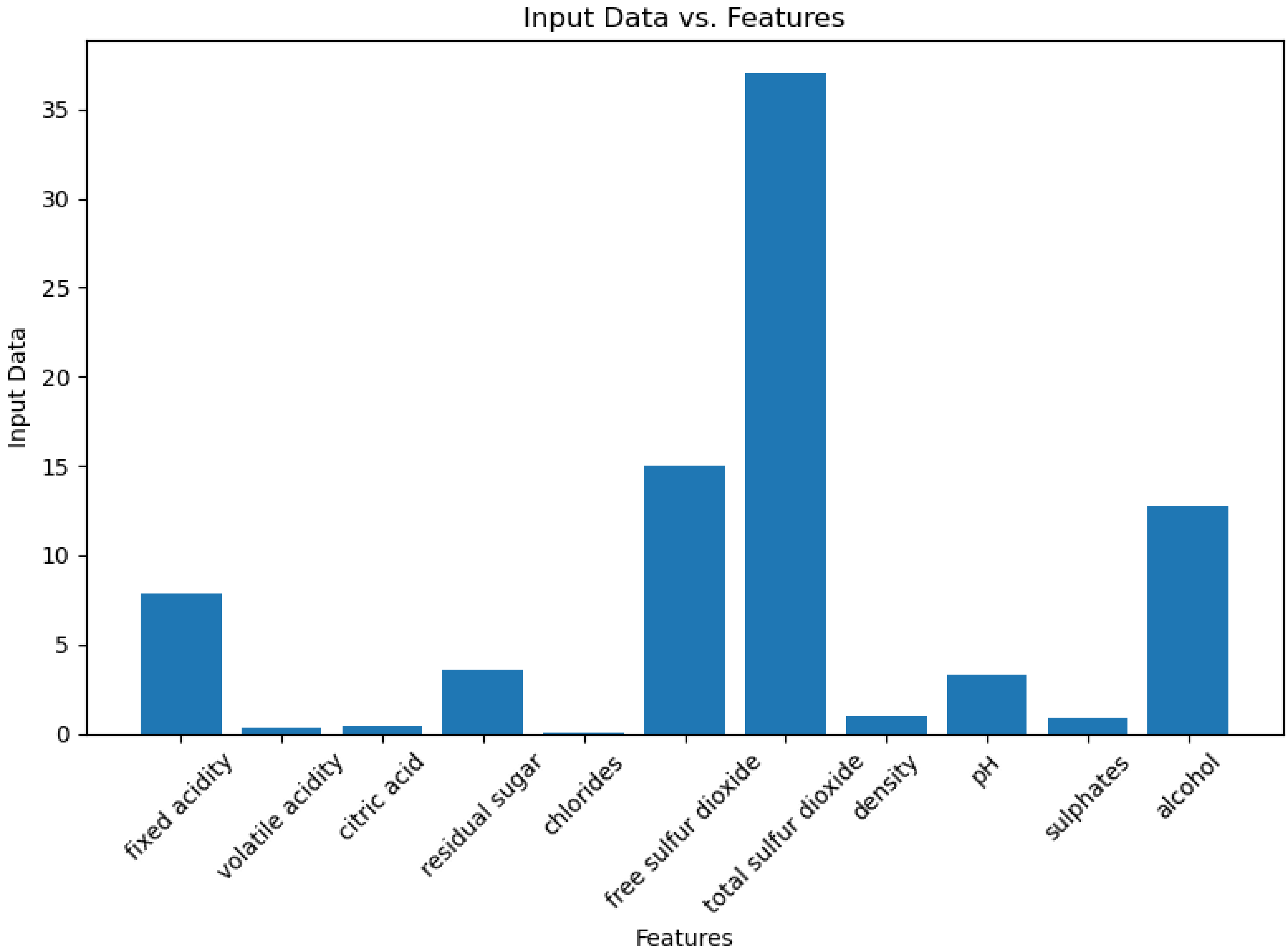
In [118]:

```
import numpy as np

input_data = (7.9,0.35,0.46,3.6,0.078,15.0,37.0,0.9973,3.35,0.86,12.8)
feature_names = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', '

# Change the input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the np array as we are predicting for one instance
input_data_reshape = input_data_as_numpy_array.reshape(1, -1)

# Assign feature names to the reshaped array
input_data_with_features = pd.DataFrame(input_data_reshape, columns=feature_names)

prediction = model.predict(input_data_with_features)
if prediction[0] == 1:
    print('Good Quality Wine')
else:
    print('Bad Quality Wine')

```

```
Good Quality Wine
```

In [119]:

```
# Plot the input data against the features
plt.figure(figsize=(8, 6))
plt.bar(feature_names, input_data_as_numpy_array)
plt.xlabel('Features')
plt.ylabel('Input Data')
plt.title('Input Data vs. Features')
plt.xticks(rotation=45)  # Rotate the x-axis labels by 45 degrees
plt.tight_layout()  # Adjust the layout to prevent label overlapping
plt.show()
```


Input Data vs. Features
Input Data
35
30
25
20
15
10
5
0
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
Features